DOI: 10.14529/jsfi210207

Optimizing Load Balance in a Parallel CFD Code for a Large-scale Turbine Simulation on a Vector Supercomputer

***Osamu Watanabe*[1], *Kazuhiko Komatsu*[2], *Masayuki Sato*[2], *Hiroaki Kobayashi*[2]**

© The Authors 2021. This paper is published with open access at SuperFri.org

A turbine for power generation is one of the essential infrastructures in our society. A turbine's failure causes severe social and economic impacts on our everyday life. Therefore, it is necessary to foresee such failures in advance. However, it is not easy to expect these failures from a real turbine. Hence, it is required to simulate various events occurring in the turbine by numerical simulations of the turbine. A multiphysics CFD code, "Numerical Turbine," has been developed on vector supercomputer systems for large-scale simulations of unsteady wet steam flows inside a turbine. To solve this problem, the Numerical Turbine code is a block structure code using MPI parallelization, and the calculation space consists of grid blocks of different sizes. Therefore, load imbalance occurs when executing the code in MPI parallelization. This paper creates an estimation model that finds the calculation time from each grid block's calculation amount and calculation performance. It proposes an OpenMP parallelization method for the load balance of MPI applications. This proposed method reduces the load imbalance by considering the vector performance according to the calculation amount based on the model. Moreover, this proposed method recognizes the need to reduce the load imbalance without pre-execution. The performance evaluation shows that the proposed method improves the load balance from 24.4 % to 9.3 %.

Keywords: turbine simulation code, MPI, OpenMP, hybrid parallelization, vector supercomputer, load balance.

Introduction

Thanks to advances in large-scale simulations, various phenomena in the real world can be reproduced more realistically by using supercomputer systems. On the other hand, there are still many problems with social infrastructures to be solved in the real world, and the impact of these issues on our society is immeasurable [1]. Therefore, there is no doubt that preventing these problems is beneficial for promoting a safe society. For example, gas and steam turbines are used to generate thermal power. Their failures will have a severe social and economic impact. Therefore, it is necessary to foresee such failures in advance. However, it is difficult to predict these failures from a real turbine [10]. Consequently, it is needed to simulate various phenomena occurring in the turbine by a numerical simulation using a computer to predict the failures. Moreover, the use of supercomputers is indispensable for simulating complex and diverse phenomena that occur in turbines.

Internally, these turbines consist of multiple stages of stator cascades and rotor cascades, and the total number of blades exceeds one thousand. It is difficult from the expense and time to realize the designs of these turbines in the short term in practical ways. To design highly reliable advanced gas turbines and steam turbines, it is necessary to concurrently solve various physical phenomena that occur in parallel with the heat flow phenomena (e.g., adhesion of fine particles to a blade, condensation of water vapor, erosion due to collision of large droplets with a blade, melting of a blade by high-temperature thermal fluid, corrosion of a blade by supercritical water, etc.). Therefore, to design a highly efficient and reliable turbine, it is necessary to develop a multiphysics CFD technology capable of numerically analyzing mathematical models simulating

[1]NEC Corporation, Tokyo, Japan
[2]Tohoku University, Sendai, Japan

this multiphysics in conjunction with governing equations of thermal flows [11, 19]. However, mutual interference of multiphysics in the turbine is due to problematic mutual interference with all heat flow fields in the turbine. For this reason, it is complicated to solve this multiphysics without thoroughly analyzing the total internal heat flow of the turbine simultaneously.

A multiphysics CFD code, "Numerical Turbine," has been developed for large-scale simulations of unsteady wet steam flow with non-equilibrium condensation inside a turbine [14]. The Numerical Turbine code has been developed on vector supercomputer systems [3, 4, 9], and the code has been optimized so that high computing performance can be obtained with vector supercomputers [7, 20]. The code can be used to analyze the unsteady wet steam flow in the final multi-stage cascade of a real steam turbine. The code also applies numerical solutions for analyzing the complex thermal flow generated inside the final stage of a steam turbine. The code incorporating these mathematical models can numerically elucidate the multiphysics interaction of the thermal flow inside the turbine. It is possible to determine in advance a catastrophic situation leading to turbine instability and blade destruction.

To accurately simulate such various phenomena into the turbine, it is essential to do a whole circumference simulation that analyzes the entire turbine. The number of computational grids to be calculated exceeds 500 million. The amount of calculation and memory used is enormous in doing the full annulus simulation. Moreover, to use analysis results practically, it is necessary to complete calculations within a required time.

The Numerical Turbine code is a block-structured CFD code, and the domain decomposition is chosen for the MPI parallelization of the Numerical Turbine code. Each MPI process is in charge of one or more grid blocks exchanging information with the neighbor ones. The difference in the number of grid points of the grid blocks causes the difference in the calculation amount of the MPI process of each grid block. Consequently, the calculation time of these MPI processes is also different. Thus, executing this code in MPI parallelization causes a load imbalance.

An MPI process with a short calculation time needs to wait for other MPI processes with a long calculation time in such a situation. Additionally, an increase in the number of MPI parallels with load imbalance adversely affects scalability. This situation indicates that the computational resources of the supercomputer are not being used effectively.

Therefore, to execute the Numerical Turbine code on a vector computer, this paper creates an estimation model that finds the calculation time from each grid block's calculation amount and calculation performance. It proposes a method for reducing the load imbalance by considering the vector performance according to the calculation amount based on the model. For parallelization of the grid blocks, thread parallelization using OpenMP is applied, and load balance is improved by hybrid parallelization. However, in the vector architecture, the length of the vectorized loop has a significant effect on the performance. Because of this, the vector performance according to the calculation amount needs to be considered for improving the load imbalance. Moreover, this proposed method recognizes the need to reduce the load imbalance without pre-execution.

The outline of this paper is as follows. In Section 1, this paper gives an overview of the Numerical Turbine code and characteristics of the code in MPI parallel execution. This paper proposes a method for reducing load imbalance using hybrid parallelization in Section 2. In Section 3, this paper discusses the performance results and concludes the paper and future work in Section 3.3.

1. Numerical Turbine Code

The Numerical Turbine code can simulate unsteady flows with wetness and shocks in actual gas and steam turbines and perform full annulus simulations of the turbines to reproduce unsteady wet-steam moist-air flows in these turbines.

1.1. Numerical Procedure

The Numerical Turbine code solves equations consisting of a mass conservation equation of steam considering momentum phase change, a momentum conservation equation, an energy conservation equation, a droplet mass conservation equation, a droplet number density conservation equation, and an equation of turbulence kinetic energy along with its specific dissipation rate as a fundamental equation of compressible flow with condensation. It is assumed that the gas-liquid two-phase flow is a homogeneous flow with a sufficiently small mass fraction of droplets. The equation of the state of wet steam and the equation of the speed of sound are calculated from the equation formulated by Ishizaka et al. [8]. The mass production rate of liquid droplets by condensation is expressed by the sum of condensation nucleation and mass growth by droplet growth based on classical condensation theory. In the Numerical Turbine, the droplet growth is further approximated by the equation with the number density of droplets as a function [8]. The condensation nucleation rate is calculated from the equation of Frenkel [12], and the growth rate of droplets is calculated from the model of Gyarmathy [6]. As the numerical solution, Roe's flux difference separation method [17] and the fourth-order compact MUSCL TVD scheme [22] are used for the spatial difference. The second precision central difference is used for the viscosity term, and the SST model [13] is used for the turbulent flow model. The lower-upper symmetric Gauss-Seidel (LU-SGS) method [23] is used for time integration.

1.2. Computational Space and Grid

As shown in Fig. 1a, the actual turbine is calculated in the grid-structured calculation space shown in Fig. 1b in the Numerical Turbine code. Figure 2 is a schematic diagram of Fig. 1b. In the figure, the stator-blade rows and the rotor-blade rows are marked as 1S, 1R, 2S, 2R, 3S, and 3R, where S is a stator-blade row and R is a rotor-blade row. A pair of a stator-blade row and a rotor-blade row is called a stage. In addition to a stator-blade row and a rotor-blade row, there are an inlet region in front of the first stage, an outlet region in back of the last stage, and intermediate regions in between neighboring blade rows. Each stator- and rotor-blade row consists of grid blocks for each blade passage. Similar to the rows, the other regions consist of grid blocks for each passage. Figure 3 illustrates a schematic diagram of the definition of the computational grid number of a grid block. In the Numerical Turbine, the axial grid number is defined as I, the circumferential grid number as J, and the radial grid number as K. The number and the grid size of these grid blocks vary depending on the rows or the regions.

1.3. Load Imbalance in MPI Parallel Execution

The computational space of the Numerical Turbine code is divided into these grid blocks by MPI parallelization. Therefore, each process in MPI parallelization handles its associated grid block. However, as described in the previous sub-section, since the grid block size is not uniform, load imbalance occurs when the Numerical Turbine code runs in MPI parallelization.

(a) Photograph of five-stage stators and rotors

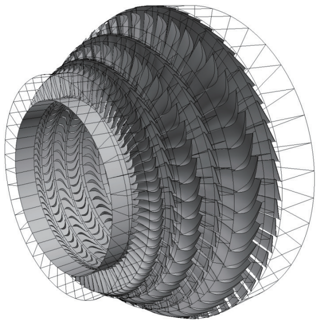
(b) Overall perspective view of computational grid frames of three-stage stators and rotors

Figure 1. Multi-stage stators and rotors of turbine

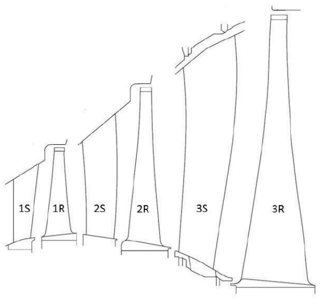

Figure 2. Schematic of the stator and rotor blades of the three stages

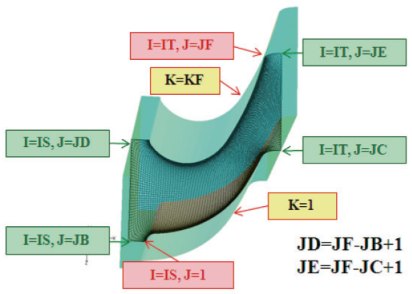

Figure 3. Calculation grid number definition per grid block

Here, the situation of this load imbalance is shown using actual simulation data. The datum is for a full annulus simulation of the first stage of a compressor, and the total number of the grid blocks is 174. Therefore, the maximum number of MPI processes is 174. Table 1 shows the number of the grid blocks and the number of the grid points in each row of the full annulus simulation data of the first stage of the compressor. As shown in this table, this datum has three types of grid points: 91×91×91, 45×91×91, and 16×91×91. In this datum, the grid blocks of Inlet 2, Rotor, Stator, and Outlet 1 have the largest number of grid points at 91×91×91. The grid blocks of Inlet 1 and Outlet 2 have the number of grid points at 45×91×91. The grid blocks of Intermediate 1 and Intermediate 2 have the smallest number of grid points at 16×91×91.

Table 1. Number of grid blocks and grid number of the full annulus data of the first stage of the compressor

		Inlet 1	Inlet 2	Rotor	Inter-mediate 1	Inter-mediate 2	Stator	Outlet 1	Outlet 2	Total number
# of blocks		16	16	32	16	16	46	16	16	174
Grid number	*I*	45	91	91	16	16	91	91	45	–
	J	91	91	91	91	91	91	91	91	–
	K	91	91	91	91	91	91	91	91	–

Figure 4 shows the cost distribution of the calculation time, the communication time, and the waiting time when the Numerical Turbine code using this simulation datum runs at 174 MPI processes. In the graph of this figure, the horizontal axis shows the rank number of each MPI process, and the vertical axis shows the execution time. The blue, orange, and gray parts respectively indicate the calculation time, the communication time, and the waiting time. As shown

		Inlet 1	Inlet 2	Rotor	Inter-mediate 1	Inter-mediate 2	Stator	Outlet 1	Outlet 2	Number of total blocks
Number of blocks		16	16	32	16	16	46	16	16	174
Computational grid number	I	45	91	91	16	16	91	91	45	-
	J	91	91	91	91	91	91	91	91	-
	K	91	91	91	91	91	91	91	91	-

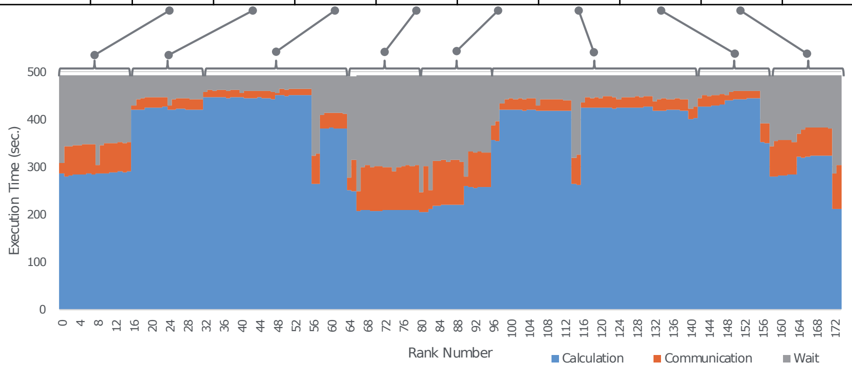

Figure 4. Cost distribution of the compressor in pure MPI and grid number of each grid block

in the figure, the MPI processes of the grid blocks with larger grid points take more time for the calculation than the MPI processes of the grid blocks with fewer grid points. Therefore, the MPI processes of the grid blocks with fewer grid points have a longer waiting time.

In parallel execution with such load imbalance, high-performance simulations with effective use of computational resources are not possible. For realizing parallel execution with effective use of computational resources, the computational load must be leveled. It is necessary to uniform calculation time by parallelizing grid blocks with a larger number of grid points for realizing such execution.

1.4. Related Work

For performing large-scale simulations, parallel execution of simulation codes is indispensable. However, to realize efficient parallel execution, it is necessary to reduce the load imbalance that occurs during parallel execution as much as possible. Research to reduce such load imbalance has been widely conducted.

Musa et al. [15] demonstrated that the workload of each process needed to be as equal as possible for achieving efficient parallelization. Their tsunami simulation program was parallelized by the domain decomposition method using MPI. In the simulation program, calculation amounts in the land areas differed from those in the sea areas. Hence, they adjusted the grid point number of each process to coincide with roughly the same calculation amount. First, the calculation amounts of each MPI process were measured with the same number of grid points per MPI process. Then the grid point number on each MPI process was adjusted to coincide with the almost equal calculation amount by using the previous measurement. As a result, they showed that the load balance in each MPI process handling the nearly even calculation amounts was better than the load balance in each MPI process with the same number of grid points. However, in the vector architecture, the length of the vectorized loop has a significant effect on the performance. Therefore, increasing and decreasing both the calculation amount and the vector length need to be considered for improving the load balance. In addition, the method of Musa et al. [15] needs to be executed in advance to equalize the amount of calculation, even though the tsunami simulation program causes static load imbalance. The method is not suitable for the Numerical Turbine code that targets various simulation data.

Simmendinger et al. [18] explained how further splitting the block into smaller parts for a block-structured CFD solver was often impractical. They mentioned that smaller MPI domains came with high overhead in terms of communication and decreased convergence rates for implicit solvers. Therefore, they implemented a hybrid parallelization model based on pthreads and MPI. Their implementation showed a significant speed-up when using more cores than domains existing in the mesh.

Giovannini et al. [5] proposed exploiting vectorization capability for serial optimization of a 3-D multi-row, multi-block CFD solver. They also proposed implementing hybrid parallelization (MPI+OpenMP) for the parallelization of the CFD solver. They described that more finely partitioning does not necessarily result in higher scalability. Furthermore, they mentioned that a high domain decomposition could have a detrimental effect on some convergence accelerating techniques (e.g., residual smoothing, multi-grinding, etc.). For these reasons, they did not implement further domain partitioning and pursued code flexibility adopting a hybrid parallelization strategy. The Numerical Turbine code is also a CFD code with a block structure that constitutes a 3-D multi-row and multi-block structure. As shown in Fig. 4, there are processes with a large ratio of communication time. Therefore, further dividing the blocks in MPI parallelization may lead to increasing the communication time and to an increase in execution time. Hence, as Simmingender et al. and Giovannini et al. mention, applying thread parallelization such as OpenMP is suitable for dividing the blocks.

Rabenseifner et al. [16] discussed that a hybrid parallelization model of MPI+OpenMP had several advantages, and the benefits include improving the load balance. They indicated that one of the significant advantages of OpenMP over MPI is the possible use of loop scheduling. They also showed that no additional programming work or data movement is required for using these loop scheduling. Moreover, they explained that simple static load imbalance at the external level (MPI) and OpenMP loop scheduling could be used as a compromise for the hybrid case. As Rabenseifner et al. [16] described, by applying hybrid parallelization to an MPI program, it is possible to realize to reduce a load imbalance of the program with easy implementation.

Based on these related studies, a way to improve the load balance of the Numerical Turbine code, a block-structured CFD code, is to apply OpenMP parallelization for dividing the grid blocks further and adjusting the workload. Thereby, the computation time among grid blocks can be equalized. However, when running a program on a vector computer, the load imbalance cannot be solved by simply equalizing the calculation amount since the computational performance depends on the loop length of the vectorized loop. Therefore, this paper proposes a method to achieve good load balance by considering the trade-off between calculation amount and the effect of vector length. A significant point to determine the number of parallelization is to perform appropriate domain dividing and mapping of computational resources among different calculation amounts and computational characteristics for each condition, such as the size and number of grid blocks and the number of blade rows in various turbine simulation. Thus, it is not realistic to execute the program in advance for each simulation condition to determine the number of parallelization for improving the load balance. Hence, the proposed method describes a way to reduce the load imbalance without pre-execution.

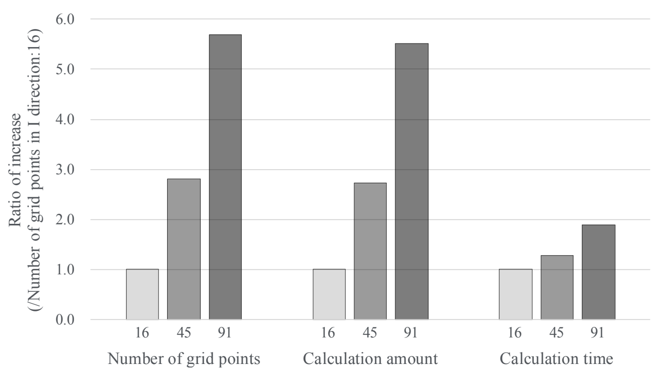

Figure 5. Ratio of calculation amount and calculation time according to the number of grid points

2. Optimizing Load Balance Using Hybrid Parallelization

As mentioned in the previous section, the computational amount of each grid block is different because the Numerical Turbine code is block-structured, and the number of grid points in each grid block is non-uniform. As a result, there is a difference in the calculation time of the MPI process in charge of each grid block. The difference causes load imbalance in MPI parallel execution of the Numerical Turbine code. This paper creates an estimation model that finds the calculation time from each grid block's calculation amount and calculation performance to reduce such load imbalance. It proposes a method of assigning threads to the MPI process in charge of each grid block based on the estimation model.

As shown in Fig. 4, it is clear that the main factor of the load imbalance is the unevenness of the calculation time. In general, the calculation time depends on the calculation amount. If the calculation time is proportional to the calculation amount, the number of threads based on the ratio of the grid points can be assigned to each MPI process. However, in vector processing, the length of the vectorized loop also greatly affects the performance. Figure 5 respectively shows the grid point ratio, calculation amount ratio, and calculation time ratio in the simulation data for a full annulus simulation of the first stage of a compressor shown in the previous section. There are three types of grid points for each grid block in this simulation data. However, these three types of grid points differ only in the number of grid points in the I-direction. Therefore, this figure shows the number of grid points in the I-direction as the number of grid points. In this figure, each number of grid points in the I-direction is 16, 45, and 91. This figure shows values normalized by the case where the number of grid points in the I-direction is 16. As shown in this figure, the calculation amount ratio is almost the same as the grid point ratio, whereas the calculation time ratio is smaller than the grid point ratio. This is because the loop length of the vectorized loop becomes longer as the number of the grid points increases so that the calculation performance by vector processing is improved.

For this reason, it is suitable for vector supercomputers to assign the number of threads based on the calculation time ratio. For finding the calculation time ratio, it is necessary to perform pre-execution. However, since the Numerical Turbine code computes simulation data of various problem sizes, it is not suitable in terms of usability to perform pre-execution for each simulation data. Therefore, this paper focuses on the fact that the computation performance is the same if the grid points are the same since the same Numerical Turbine code is used to calculate different simulation data. It creates the estimation model to obtain the calculation time from each grid block's calculation amount and calculation performance. This paper finds the number of threads assigned to each MPI process without pre-execution based on this model. The following describes the details of the model.

- The following relational equation estimates the calculation time ratio as:

$$calculation\ time\ ratio = \frac{calculation\ amount\ ratio}{calculation\ performance\ ratio}. \tag{1}$$

- The calculation amount ratio can be found in advance because the ratio is equivalent to the grid point ratio.
- The calculation performance can be identified according to the number of grid points because the same Numerical Turbine code is used even for different simulation data. Therefore, the calculation performance ratio can be determined in advance.

Hence, the ratio of the number of threads according to the calculation time ratio can be found in advance by the following equation. According to this ratio, it is possible to determine the number of threads assigned to each MPI process without pre-execution. The ratio of the number of threads is written as:

$$thread\ number\ ratio = \frac{grid\ point\ ratio}{calculation\ performance\ ratio}. \tag{2}$$

The following describes the details of the proposed method based on the above model. To obtain the calculation time ratio without pre-execution, it is necessary to mathematize the relationship in Eq. 1. Hence, the relationship between the number of grid points and the calculation performance in the Numerical Turbine code was clarified by a performance evaluation.

Figure 6 shows the relationship between the number of grid points and the calculation performance in the Numerical Turbine code. To find this relationship, varying the calculation performance was confirmed according to vary the number of grid points using an evaluation code that extracted the calculation parts of the Numerical Turbine code. The calculation parts do not include the communication parts. In this evaluation, only the grid points in the I-direction were varied because the number of grid points in the J- and K-directions of the grid block is the same for all grid blocks.

In this figure, the horizontal axis shows the number of grid points in the I-direction, and the vertical axis shows the calculation performance ratio. The calculation performance ratio on the horizontal axis is a normalized value for the calculation performance when the number of grid points in the I-direction is 10. The number of grid points in the I-direction used is at least two-digit or more in actual simulations. As this figure shows, the calculation performance ratio increases as the number of grid points increases. The relationship is not proportional, and the calculation performance ratio increases significantly until the number of grid points in the I-direction changes from two digits to three digits. When the number of grid points in the I-direction is about 10, the loop length of the vectorized loops is short, and the vector arithmetic unit cannot fully exhibit its calculation performance. As the number of grid points in the I-direction increases, the loop length increases, and the vector arithmetic unit comes to show its high calculation performance.

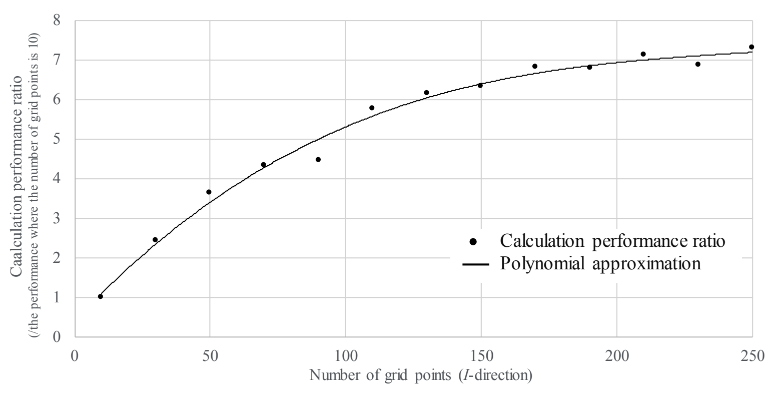

Figure 6. Relationship between the number of grid points and the calculation performance ratio

Table 2. Values of the polynomial coefficients

Coefficient	Value
a	3.62772E-07
b	–0.000272569
c	0.072582348
d	0.412525894

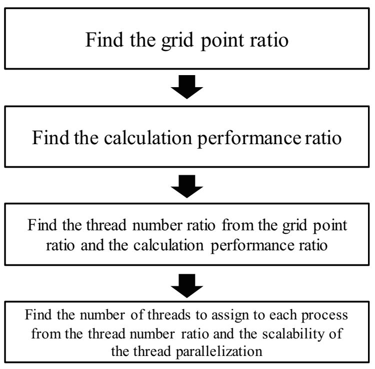
Find the grid point ratio
Find the calculation performance ratio
Find the thread number ratio from the grid point ratio and the calculation performance ratio
Find the number of threads to assign to each process from the thread number ratio and the scalability of the thread parallelization

Figure 7. Procedure of finding the number of threads to assign to each process

However, as described in Section 1.4, the calculation amount differs depending on the conditions in the actual simulation. Therefore, it is not appropriate to obtain the calculation performance ratio under various conditions from the graph in terms of accuracy and convenience. Hence, an approximate curve is obtained from the measurement data shown in Fig. 6. In the actual simulation, the calculation performance ratio is obtained from an equation of the approximate curve. Equation 3 is the equation of this approximate curve. The equation of an approximate curve can express the relationship between the number of the grid points in the I-direction and the calculation performance ratio shown in Fig. 6.

$$y = a * x^3 + b * x^2 + c * x + d, \tag{3}$$

where x is the number of grid points in the I-direction, and a, b, c, and d are the polynomial coefficients. Table 2 shows the values of each polynomial coefficient.

Figure 7 shows the procedure of the proposed method, which is roughly composed of four steps. The first step is to calculate each grid point ratio based on the maximum number of grid points (g_N) as shown in Tab. 3, where $g_1 < g_2 < g_3 \ldots < g_n < \ldots < g_{N-2} < g_{N-1} < g_N$ $(1 \leq n \leq N)$. As shown in this table, the grid point ratio at grid point g_n is g_n/g_N.

The second step is to calculate each calculation performance ratio based on the calculation performance of the maximum number of grid points (y_N) as shown in Tab. 3. As shown in this table, the calculation performance ratio at grid point g_n is y_n/y_N. Here, the calculation performance (y_n) is obtained from Fig. 6.

Table 3. Three kinds of ratios

# of grid points	g_1	g_2	g_3	$\cdots$	g_n	$\cdots$	g_{N-2}	g_{N-1}	g_N
Grid point ratio	$\frac{g_1}{g_N}$	$\frac{g_2}{g_N}$	$\frac{g_3}{g_N}$	$\cdots$	$\frac{g_n}{g_N}$	$\cdots$	$\frac{g_{N-2}}{g_N}$	$\frac{g_{N-1}}{g_N}$	$\frac{g_N}{g_N}$ $(=1)$
Calculation performance ratio	$\frac{y_1}{y_N}$	$\frac{y_2}{y_N}$	$\frac{y_3}{y_N}$	$\cdots$	$\frac{y_n}{y_N}$	$\cdots$	$\frac{y_{N-2}}{y_N}$	$\frac{y_{N-1}}{y_N}$	$\frac{y_N}{y_N}$ $(=1)$
Thread number ratio	$\frac{G_1}{Y_1}$	$\frac{G_2}{Y_2}$	$\frac{G_3}{Y_3}$	$\cdots$	$\frac{G_n}{Y_n}$	$\cdots$	$\frac{G_{N-2}}{Y_{N-2}}$	$\frac{G_{N-1}}{Y_{N-1}}$	$\frac{G_N}{Y_N}$ $(=1)$

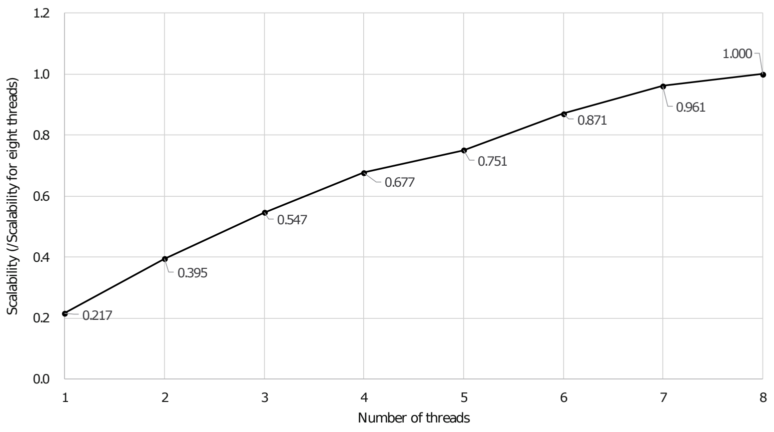

Figure 8. Scalability of the thread parallelization of the calculation part

At this point, since the grid point ratio and the calculation performance ratio have been obtained, the calculation time ratio can be obtained by dividing the grid point ratio by the calculation performance ratio at each number of grid points. In the third step, the calculation time ratio is used as the ratio of the number of threads assigned to each grid block because the ratio of the number of threads corresponds to the calculation time ratio. As shown in Tab. 3, $G_n = \frac{g_n}{g_N}$ and $Y_n = \frac{y_n}{y_N}$.

The thread number ratios obtained in Tab. 3 are real numbers less than or equal to 1.0. Since each number of threads is an integer value, it is necessary to determine each number of threads by converting the ratio into an integer value considering performance. Hence, in the final step, each number of threads is determined based on the scalability of thread parallelization in the calculation part of the Numerical Turbine code. Here, the scalability of thread parallelization in the calculation part is obtained as the ratio of the number of threads with the maximum number of the scalability as 1.0. Figure 8 shows the scalability of thread parallelization in the calculation part. This figure identifies the number of threads of the closest scalability to the ratio of the number of threads obtained in Tab. 3. Then, the number of threads is assigned to the grid block of the corresponding number of grid points. Thereby, it is possible to obtain the number of threads according to the actual performance.

Through these four steps, it is possible to determine in advance the number of threads to be assigned to each MPI process based on the ratio corresponding to the calculation time ratio.

Table 4. Specification of Vector Engine: Type 10AE

Peak performance of core (GFLOPS)	304
Number of cores per VE	8
Peak performance of VE (TFLOPS)	2.43
Memory bandwidth of VE (TB/s)	1.35
Cache capacity of VE (MB)	16
Memory capacity of VE (GB)	48

Table 5. Software environment of SX-Aurora TSUBASA

NEC Fortran compiler for VE	nfort (NFORT) 3.0.8
NEC MPI	NEC MPI 2.10.0

3. Results and Discussions

This section evaluates the effectiveness of the proposed method to improve the load balance of the Numerical Turbine code by using actual simulation data.

3.1. Input Data Set

This experiment uses the full annulus data of the first stage of the compressor as the input dataset. Table 1 shows the number of grid blocks in each row of full annulus data for the first stage of the compressor and the number of grid points in the grid blocks. As shown in the table, there are a total of 174 blocks and three types of grid points: 91×91×91, 45×91×91, and 16×91×91. The total number of grid points is about 100 million. The number of iterations is 1,000, which is the minimum number required for performance analysis. The total number of grid blocks is 174, so the maximum number of MPI processes is 174.

3.2. Computing Environment Setup

A supercomputer used for this evaluation is the modern vector supercomputer system called SX-Aurora TSUBASA, which was released in 2017 [9, 21], for evaluating the performance of the Numerical Turbine. As described in the previous section, the Numerical Turbine is well optimized for vector architectures. Therefore, to accurately evaluate the overall performance, it is suitable for the performance evaluation on a vector supercomputer. Here, an overview of SX-Aurora TSUBASA is described.

SX-Aurora TSUBASA architecture contains the Vector Engine (VE) and Vector Host (VH), as shown in Fig. 9. The VE executes complete applications, and the VH mainly provides OS functions for connected VEs. The VE consists of one vector processor with eight vector cores, using High Bandwidth Memory modules (HBM2) for uppermost memory bandwidth. The implementation of one CPU LSI with six HBM2 memory modules leads to high memory bandwidth. The VE is connected to the VH, a standard x86/Linux node, through PCIe. This architecture executes an entire application on the VE and the OS on the VH [2].

As shown in Tab. 4, the peak performances of a vector core and a VE are 304 Gflop/s and 2.43 Tflop/s, respectively, and the memory bandwidth of a VE is 1.35 TB/s. Figure 10 shows the configuration of one VH that contains eight VEs, two InfiniBand interfaces, and two Xeon processors. Moreover, SX-Aurora TSUBASA can compose an extensive system by connecting the VHs via the InfiniBand switch. As mentioned in the above explanation, the number of cores per VE of SX-Aurora TSUBASA is eight. Therefore, the maximum number of threads assigned to each MPI process in hybrid parallelization is eight. Regarding the software, Tab. 5 lists the environment of SX-Aurora TSUBASA used in this evaluation.

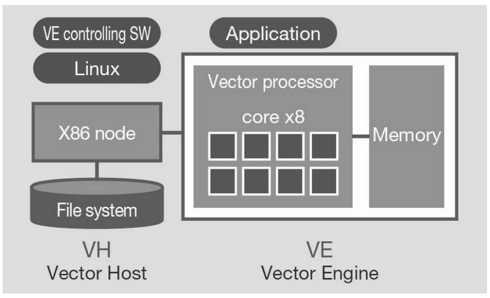

Figure 9. Architecture of SX-Aurora TSUBASA

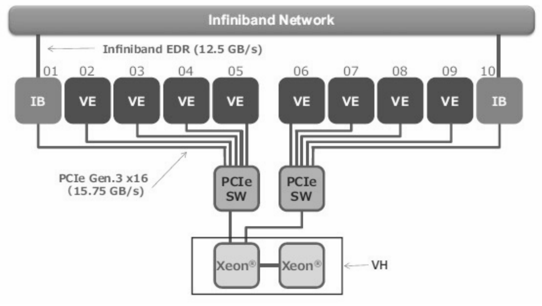

Figure 10. Configuration of SX-Aurora TSUBASA

3.3. Assigning Threads to MPI Processes

This section obtains the number of threads assigning to each MPI process following the method proposed in Section 2 using the full annulus data of the first stage of the compressor and validates the effectiveness of the proposed method.

Each grid point ratio based on the maximum number of grid points is calculated as the first step. As described in Section 3.1, the full annulus data of the first stage of the compressor comprises three types of grid blocks with grid points in the I-direction of 16, 45, and 91. Table 6 is obtained by calculating the grid point ratio according to Tab. 3.

Table 6. Three kinds of ratios in the full annulus data of the first stage of the compressor

# of grid points	16	45	91
Grid point ratio	0.176 $(=\frac{16}{91})$	0.495 $(=\frac{45}{91})$	1 $(=\frac{91}{91})$
Calculation performance ratio	0.299 $(=\frac{1.506}{5.033})$	0.628 $(=\frac{3.260}{5.033})$	1 $(=\frac{5.033}{5.033})$
Thread number ratio	0.588 $(=\frac{0.176}{0.299})$	0.788 $(=\frac{0.495}{0.628})$	1 $(=\frac{1}{1})$

As the second step, according to Tab. 3, the calculation performance ratio is obtained based on the calculation performance of the maximum number of grid points. Here, the calculation performance corresponding to each number of grid points is obtained from Fig. 6. Table 6 shows the results of the calculated calculation performance ratios.

The ratio of the number of grid points and the calculation performance ratio in the full annulus data of the first stage of the compressor has been obtained. As the third step, the ratio of the number of threads in each number of grid points is obtained based on Tab. 3. Table 6 shows the ratio of the number of threads obtained.

As the final step, to determine the number of threads in a way that takes performance into account, Fig. 8 is used to find the number of threads with the closest scalability to the ratio of the number of threads obtained in Tab. 6. The ratio of the number of threads in the number of grid points 16 is 0.564. The scalability nearest to this number is 0.547 from Fig. 8, and there are three corresponding threads. Similarly, in the number of grid points 45, the ratio of the number of threads is 0.774, and the scalability of the nearest neighbor to this number is 0.751. Thus, there are five corresponding threads. Since the number of grid points 91 is the maximum number of grid points, the maximum number of threads is assigned, which is eight. The final number of threads obtained is as shown in Tab. 7.

Table 7. The number of threads obtained by the proposed method

# of grid points	16	45	91
Ratio of the number of threads	0.588	0.788	1
The closest scalability to the above ratios	0.547	0.751	1
# of threads assigning to each MPI process	3	5	8

This section makes a comparison between hybrid parallelization based on the proposed method and the actual calculation time. Table 8 shows the number of threads obtained based on the actual calculation time. Compared to Tab. 7, the number of threads assigned to the grid block with 45 grids is four in the case based on the proposed method, while it is five in the case based on the actual calculation time. On the other hand, the number of threads is the same for the other grids. Therefore, the proposed method can obtain almost the same number of threads as the number of threads obtained based on the actual calculation time. Moreover, the execution time is 144.9 seconds in the case based on the proposed method, and is 144.6 seconds in the case based on the actual calculation time. Hence, the proposed method can obtain the execution time equivalent to hybrid parallelization based on the actual calculation time without pre-execution.

Table 8. The number of threads decided by the actual calculation time

# of grid points	16	45	91
Calculation time (sec.)	216.0	278.1	405.5
Ratio of the number of threads	0.533	0.686	1
The closest scalability to above ratios	0.547	0.677	1
# of threads assigning to each MPI process	3	4	8

Figure 11 shows the execution results by assigning the obtained number of threads to the MPI process in charge of the grid block corresponding to each grid point. To clear the efficiency of the load balance improvement, the vertical axis of this figure is set to the same scale as in Fig. 4. As can be seen from the comparison between Figs. 4 and 11, the hybrid parallelization applying the number of threads obtained by the proposed method equalizes the computation time and achieves good load balance.

For verifying the improvement effect of the load balance by the proposed method, the improvement in the variation between the pure MPI and the proposed hybrid parallelization is confirmed. The coefficient of variation is used to indicate the degree of variation. Table 9 shows the coefficient of variation for the calculation time in the pure MPI and the proposed hybrid parallelization. This table shows that the load balance is improved from 24.4 % to 9.3 % by the hybrid parallelization using the proposed method.

For verifying the efficiency of the computational resource utilization, the proposed hybrid parallelization is compared with the maximum number of threads assigned to all processes. The maximum number of threads assigned to each process is eight because the maximum number of cores per VE is eight, as shown in Tab. 4. Table 10 shows the execution time and the number of cores used for the execution in both cases. As this table shows, the proposed hybrid

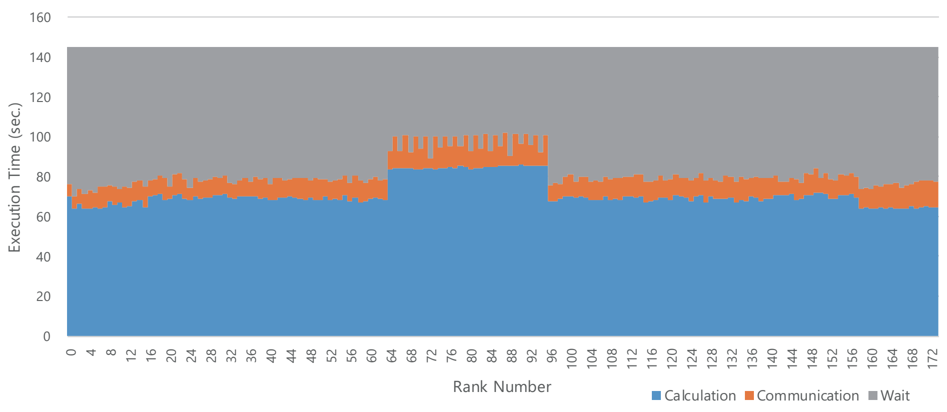

Figure 11. Cost distribution of the compressor in hybrid parallelization reducing load imbalance

Table 9. Coefficient of variation (CV) of the calculation time in each case

	Calculation Time (sec.)			
	minimum	maximum	average	CV
Pure MPI	205.83	441.99	348.42	24.4 %
Proposed hybrid parallelization	63.74	85.92	71.39	9.3 %

parallelization has the same execution time using 82 % of the computational resources of the maximum number of threads assigned to all processes. This section demonstrates the above verification results to reduce the load imbalance by hybrid parallelization, assigning the number of threads in advance based on the proposed method. As a result, the Numerical Turbine code can calculate simulations faster with efficient use of computational resources.

Table 10. Execution time and computational resources used

	Execution time (sec.)	# of cores used
Proposed hybrid parallelization	144.9	1,136
Hybrid parallelization using the maximum number of threads	145.4	1,392

Conclusions

This paper demonstrates that a way to improve the load balance of the Numerical Turbine code, a block-structured CFD code, is to apply OpenMP parallelization for dividing the grid blocks further and adjusting the workload. Thereby, the calculation time among grid blocks can be equalized. For executing the Numerical Turbine code on a vector computer, this paper creates an estimation model that finds the calculation time from each grid block's calculation amount and calculation performance. This proposed method reduces the load imbalance by considering the calculation amount and the effect of vector length based on the model. Moreover, the Numerical Turbine code has a static load imbalance and treats various simulation data. Hence, this proposed method can find suitable numbers of threads to reduce the load imbalance without pre-execution. As a result, the proposed method can improve the load balance from

24.4 % to 9.3 %, and realize 3.32 times speed-up of the Numerical Turbine code with effective usage of the computational resources.

As mentioned above, the Numerical Turbine code treats various simulation data. Some of these data have more grid blocks and more stages than the full annulus data of the first stage of the compressor used for this evaluation. Our future work will verify the effectiveness of the proposed method in these various simulation data treated by the Numerical Turbine code. In addition, the proposed method has been developed for vector supercomputers. This method may be effective for modern scalar supercomputers because the SIMD mechanism has been strengthened, and multi-core has been advanced in such scalar supercomputers. Therefore, our future work will also verify the effectiveness of this method for scalar supercomputers and improve it to a general-purpose method.

Acknowledgements

This research was supported in part by MEXT as “Next Generation High-Performance Computing Infrastructures and Applications R&D Program,” entitled “R&D of A Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications.” The authors thank Satoru Yamamoto, Takashi Furusawa, and Hironori Miyazawa of Tohoku University for their fruitful discussions and variable comments.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. Society 5.0. `https://www8.cao.go.jp/cstp/english/society5_0/index.html`, accessed: 2021-07-02

2. Vector Supercomputer SX Series SX-Aurora TSUBASA. `https://www.nec.com/en/global/solutions/hpc/sx/docs/SX-Aurora_e.pdf`, accessed: 2021-06-13

3. Egawa, R., Komatsu, K., Isobe, Y., *et al.*: Performance and power analysis of SX-ACE using HP-X benchmark programs. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). pp. 693–700. IEEE Computer Society (2017). `https://doi.org/10.1109/CLUSTER.2017.65`

4. Egawa, R., Fujimoto, S., Yamashita, T., *et al.*: Exploiting the Potentials of the Second Generation SX-Aurora TSUBASA. In: 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS). pp. 39–49. IEEE (2020). `https://doi.org/10.1109/PMBS51919.2020.00010`

5. Giovannini, M., Marconcini, M., Arnone, A., Dominguez, A.: A Hybrid Parallelization Strategy of a CFD Code for Turbomachinery Applications. In: 11th European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, ETC 2015, Madrid, Spain, March 23-27, 2015 (2015)

6. Gyarmathy, G.: Zur Wachstumsgeschwindigkeit kleiner Flüssigkeitstropfen in einer übersättigten Atmosphäre. Zeitschrift für angewandte Mathematik und Physik ZAMP 14(3), 280–293 (1963). `https://doi.org/10.1007/BF01601066`

7. Hougi, Y., Komatsu, K., Watanabe, O., *et al.*: A hierarchical wavefront method for LU-SGS on modern multi-core vector processors. In: 32nd International Conference on Parallel Computational Fluid Dynamics (2020)

8. Ishizaka, K.: A High-Resolution Numerical Method for Transonic Non-Equilibrium Condensation Flow through a Steam Turbine Cascade. Proc. of the 6th ISCFD, 1995 1, 479–484 (1995)

9. Komatsu, K., Momose, S., Isobe, Y., *et al.*: Performance Evaluation of a Vector Supercomputer SX-Aurora TSUBASA. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 685–696. IEEE (2018). `https://doi.org/10.1109/SC.2018.00057`

10. Komatsu, K., Miyazawa, H., Yiran, C., Sato, M., Furusawa, T., Yamamoto, S., Kobayashi, H.: Detection of machinery failure signs from big time-series data obtained by flow simulation of intermediate-pressure steam turbines (2021)

11. Lindner, F., Totounferoush, A., Mehl, M., *et al.*: ExaFSA: Parallel Fluid-Structure-Acoustic Simulation. In: Software for Exascale Computing - SPPEXA 2016-2019. Lecture Notes in Computational Science and Engineering, vol. 136, pp. 271–300. Springer (2020). `https://doi.org/10.1007/978-3-030-47956-5_10`

12. MacDougall, F.H.: Kinetic Theory of Liquids. By J. Frenkel. The Journal of Physical and Colloid Chemistry 51(4), 1032–1033 (1947). `https://doi.org/10.1021/j150454a025`

13. Menter, F.R.: Two-equation eddy-viscosity turbulence models for engineering applications. AIAA Journal 32(8), 1598–1605 (1994). `https://doi.org/10.2514/3.12149`

14. Miyake, S., Koda, I., Yamamoto, S., *et al.*: Unsteady Wake and Vortex Interactions in 3-D Steam Turbine Low Pressure Final Three Stages. Turbo Expo: Power for Land, Sea, and Air, vol. Volume 1B: Marine; Microturbines, Turbochargers and Small Turbomachines; Steam Turbines (2014). `https://doi.org/10.1115/GT2014-25491`

15. Musa, A., Watanabe, O., Matsuoka, H., *et al.*: Real-time tsunami inundation forecast system for tsunami disaster prevention and mitigation. Journal of Supercomputing 74(7), 3093–3113 (2018). `https://doi.org/10.1007/s11227-018-2363-0`

16. Rabenseifner, R., Hager, G., Jost, G.: Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, Weimar, Germany, Feb. 18-20, 2009. pp. 427–436. IEEE (2009). `https://doi.org/10.1109/PDP.2009.43`

17. Roe, P.L.: Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. J. Comput. Phys. 135(2), 250–258 (1997). `https://doi.org/10.1006/jcph.1997.5705`

18. Simmendinger, C., Kuegeler, E.: Hybrid Parallelization of a Turbomachinery CFD Code: Performance Enhancements on Multicore Architectures pp. 14–17 (2010)

19. Soga, T., Musa, A., Shimomura, Y., *et al.*: Performance evaluation of NEC SX-9 using real science and engineering applications. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. pp. 1–12. ACM (2009). https://doi.org/10.1145/1654059.1654088

20. Watanabe, O., Hougi, Y., Komatsu, K., *et al.*: Optimizing memory layout of hyperplane ordering for vector supercomputer SX-Aurora TSUBASA. In: 2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC), Denver, CO, USA, Nov. 18, 2019. pp. 25–32. IEEE (2019). https://doi.org/10.1109/MCHPC49590.2019.00011

21. Yamada, Y., Momose, S.: Vector engine processor of NEC's brand-new supercomputer SX-Aurora TSUBASA. In: International symposium on High Performance Chips (Hot Chips2018) (2018)

22. Yamamoto, S., Daiguji, H.: Higher-order-accurate upwind schemes for solving the compressible Euler and Navier-Stokes equations. Computers & Fluids 22(2), 259–270 (1993). https://doi.org/10.1016/0045-7930(93)90058-H

23. Yoon, S., Jameson, A.: Lower-upper Symmetric-Gauss-Seidel method for the Euler and Navier-Stokes equations. AIAA Journal 26(9), 1025–1026 (1988). https://doi.org/10.2514/3.10007